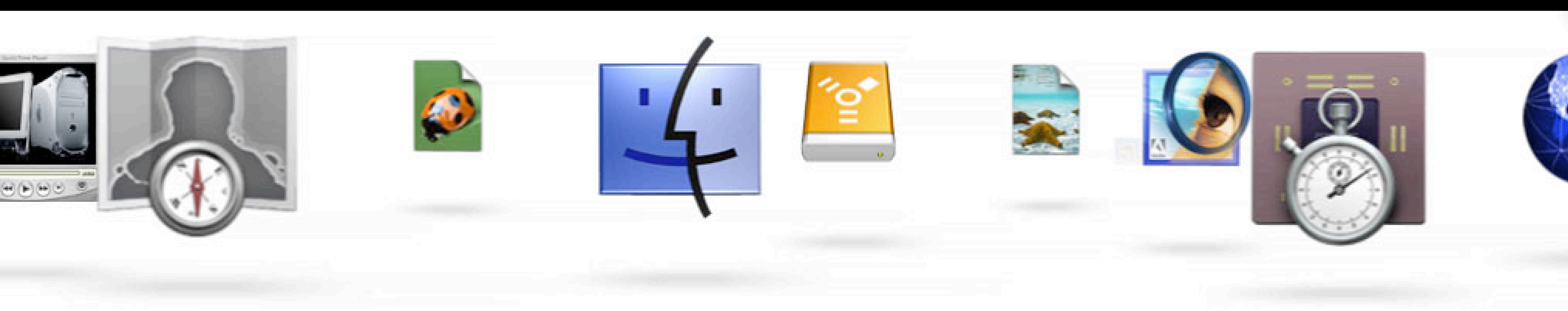
OpenGL: Advanced 3D

Session 513



OpenGL: Advanced 3D

Geoff Stahl
Apple OpenGL

Simon Green
NVIDIA

Overview

- What’s new with Jaguar?
- Wolfman Walkthrough
- Q&A

New Features

- Optimizations
  - glReadPixels
  - glCopyTexSubImage
  - glDrawPixels
  - Display Lists
  - Image Processing
  - Vertex Arrays
- Surface to Texture
- Extensions
  - 64 extensions supported
  - 30 new extensions

Programmability

- **ARB_vertex_program**
    - Adds vertex programming functionality to all software and hardware implementations
    - Support in hardware for ATI Radeon 8500, NVIDIA GeForce 3 and GeForce 4 Ti
    - All other hardware supported via an optimized software implementation
- **NV_texture_shader, NV_texture_shader2, NV_texture_shader3**
- **ATI_fragment_program**

Vertex Array

- **APPLE_vertex_array_range**
  - Allows GPU to DMA vertex arrays
  - Supported by ATI Radeon, ATI Radeon 8500, all NVIDIA. Requires hardware TCL
- **APPLE_vertex_array_object**
  - Allows vertex array state such as data pointers and vertex array range information, similar to how textures are handled
  - Supported by ATI Radeon, ATI Radeon 8500, all NVIDIA

Performance

- **APPLE_texture_range**
  - Allows texture storage hints and specification of memory range for texture data
  - Supported by ATI Radeon, ATI Radeon 8500, all NVIDIA
- **APPLE_fence**
  - Puts a fence (or token) in command stream to allow synchronization
  - All renderers supported

Texture Extensions

- ARB_texture_mirrored_repeat
- ARB_texture_env_crossbar
- ATI_texture_mirror_once
- SGIX_depth_texture
- SGIX_shadow

Rendering Extensions

- ARB_multisample
- EXT_secondary_color
- EXT_fog_coord
- EXT_draw_range_elements
- EXT_stencil_wrap
- NV_fog_distance
- NV_multisample_filter_hint
- NV_depth_clamp

Others

- Pixel Transfer
  - **ARB_imaging, ATI_blend_equation_separate, ATI_blend_weighted_minmax, NV_blend_square**
- Point Parameters
  - **ARB_point_parameters, NV_point_sprite**

Anatomy of the Wolfman

Simon Green, NVIDIA

Advanced Fur Rendering Using OpenGL Programmable Shading

NVIDIA Demo Team Unofficial Motto

"We Make The Marketing Lies Come True"

Overview

- Wolfman—one of four GeForce 4 launch demos
- Created to showcase the performance and programmability of GeForce 4
- Demonstrates volumetric fur rendering on a fully animated character model
- Animated using vertex shaders
- Per-pixel anisotropic lighting using pixel shaders
- Self-shadowing using shadows maps
- Runs using OpenGL with NVIDIA extensions
- How did we do it?

Why Fur?

- Lots of things in the real world are fuzzy
- Rendering fur is hard
- Fur is something people hadn’t seen before in real time

Rendering Fur

- Two basic methods to render fur
  - Geometric
    - Each individual hair strand is a curve (lines)
  - Volumetric
    - Fur is represented using volume texture (images)
- Hardware can’t render 3 million individual hairs per frame (yet)
- So we use the second method

A Brief History of Fur Rendering

Rendering Fur With Three Dimensional Textures

Kajiya and Kay, Siggraph 1989

- Represented fur density using 3D volume texture —“texels”
- Lit hairs based on tangent direction
- Furry teddy bear image
- Rendered on network of IBM mainframes
  - 16 processors
- 2 hours for one frame

A Brief History of Fur Rendering

Real-Time Fur Over Arbitrary Surfaces

Jed Lengyel, Microsoft Research 2001

- Introduced concept of “shell and fin” rendering
- Concentric shells approximate volume
- “Fins” improve silhouette edges
- Used “lapped textures” to cover surface
- Furry bunny rabbit
- 5000 faces
- Ran at 12 frames per second on GeForce DDR

Demo Concept

- Teddy Bears and Bunnies aren't our style, so . . . .

Demo

NVIDIA Wolfman

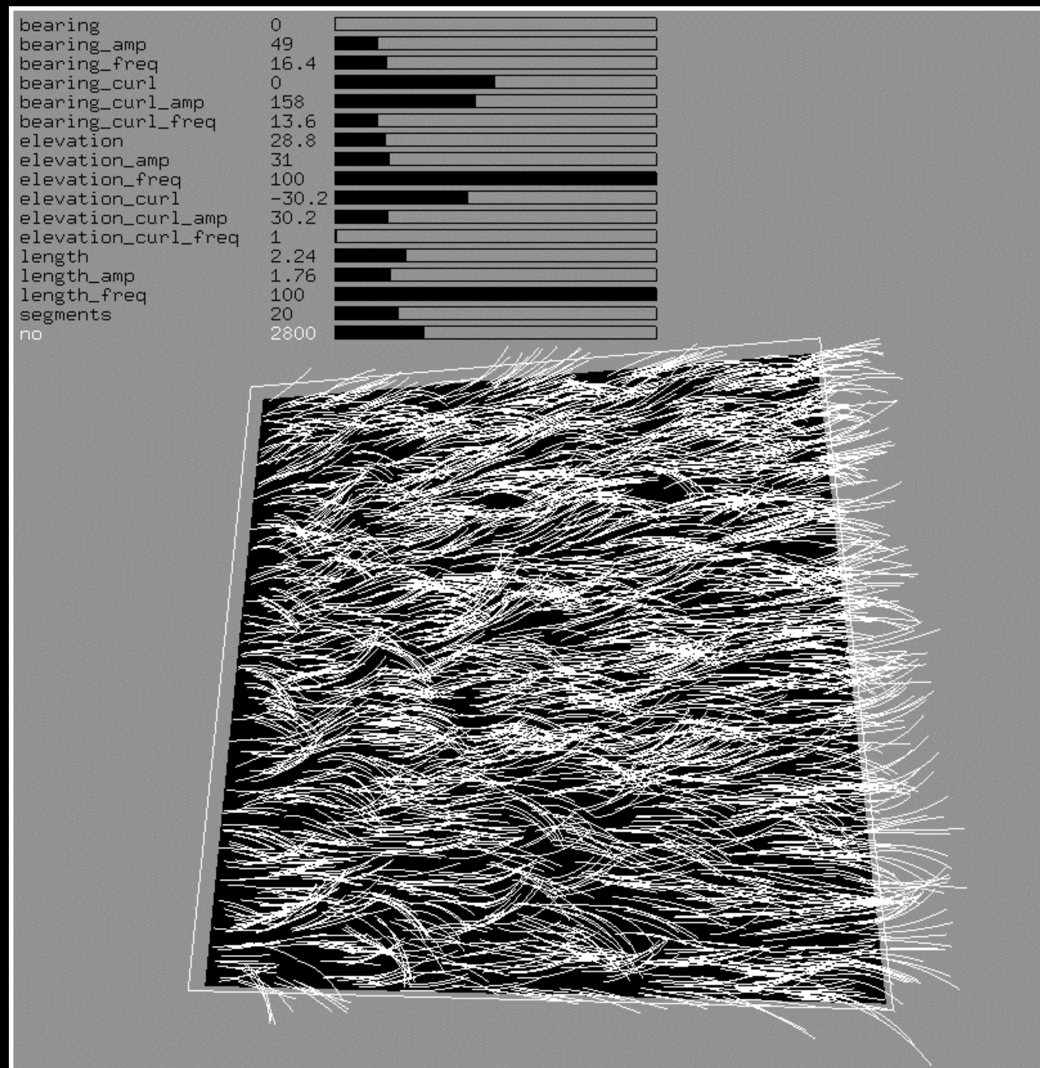
Wolfman Demo Statistics

- 100,000 polygons/frame
- Runs at around 30 frames/second

Rendering Fur With Shells and Fins

- Generate concentric “shells’ by scaling base skin mesh along vertex normal
- Each shell is textured with a different 2D texture that represents a slice of the fur geometry
- The layers are blended together to produce the illusion of a semi-transparent furry volume
- Lower layers are shaded darker to simulate self-shadowing of fur
- We used 8 layers

Fur Design Tool

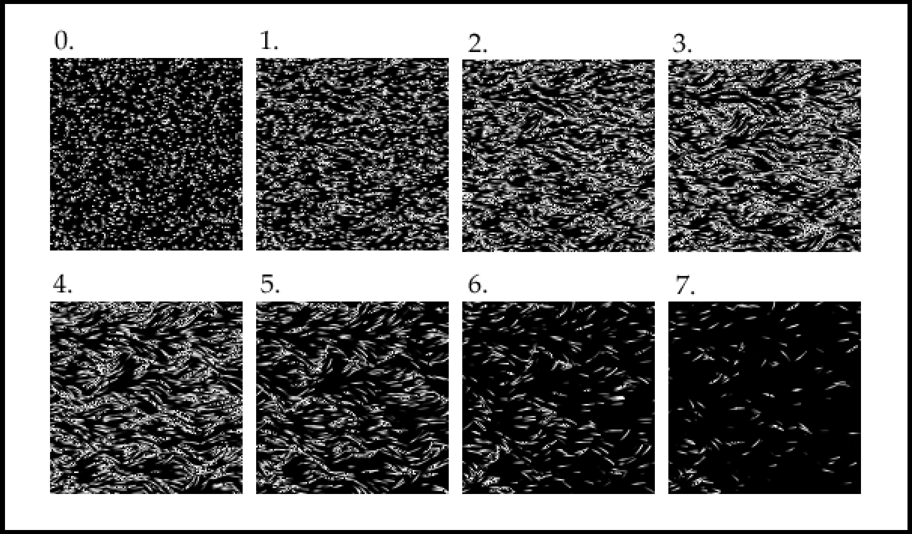


Fur Design Tool

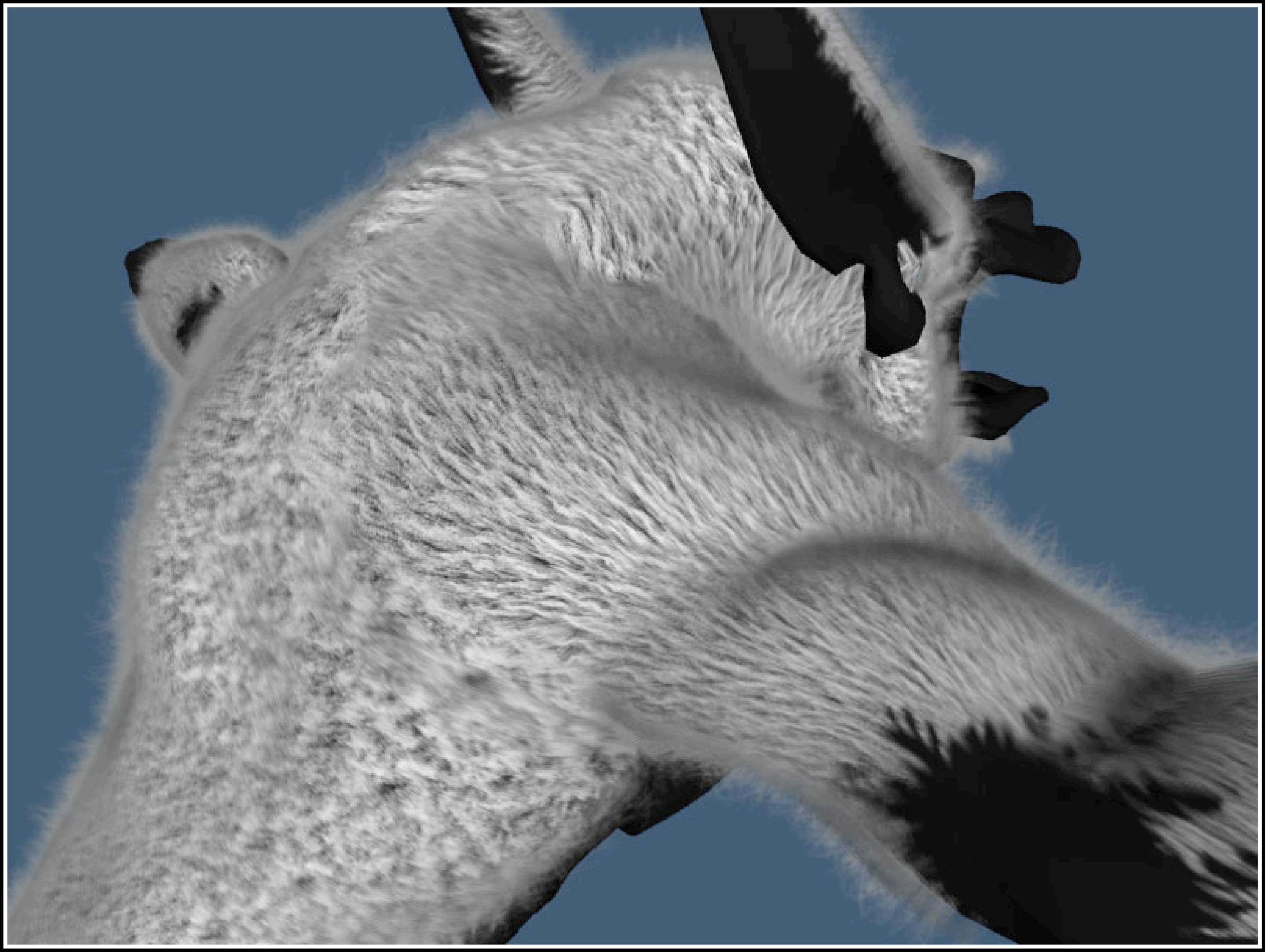
- Custom in-house tool
- Provides simple UI for designing fur textures
- Fur geometry is defined using a particle system based on spherical coordinates
- Previews hairs using line strips
- Allows direction, curliness to be tweaked using sliders
- Once you're happy, it renders (e.g., voxelizes) the geometry into a volume texture
- Writes out texture files to disk

Fur Textures

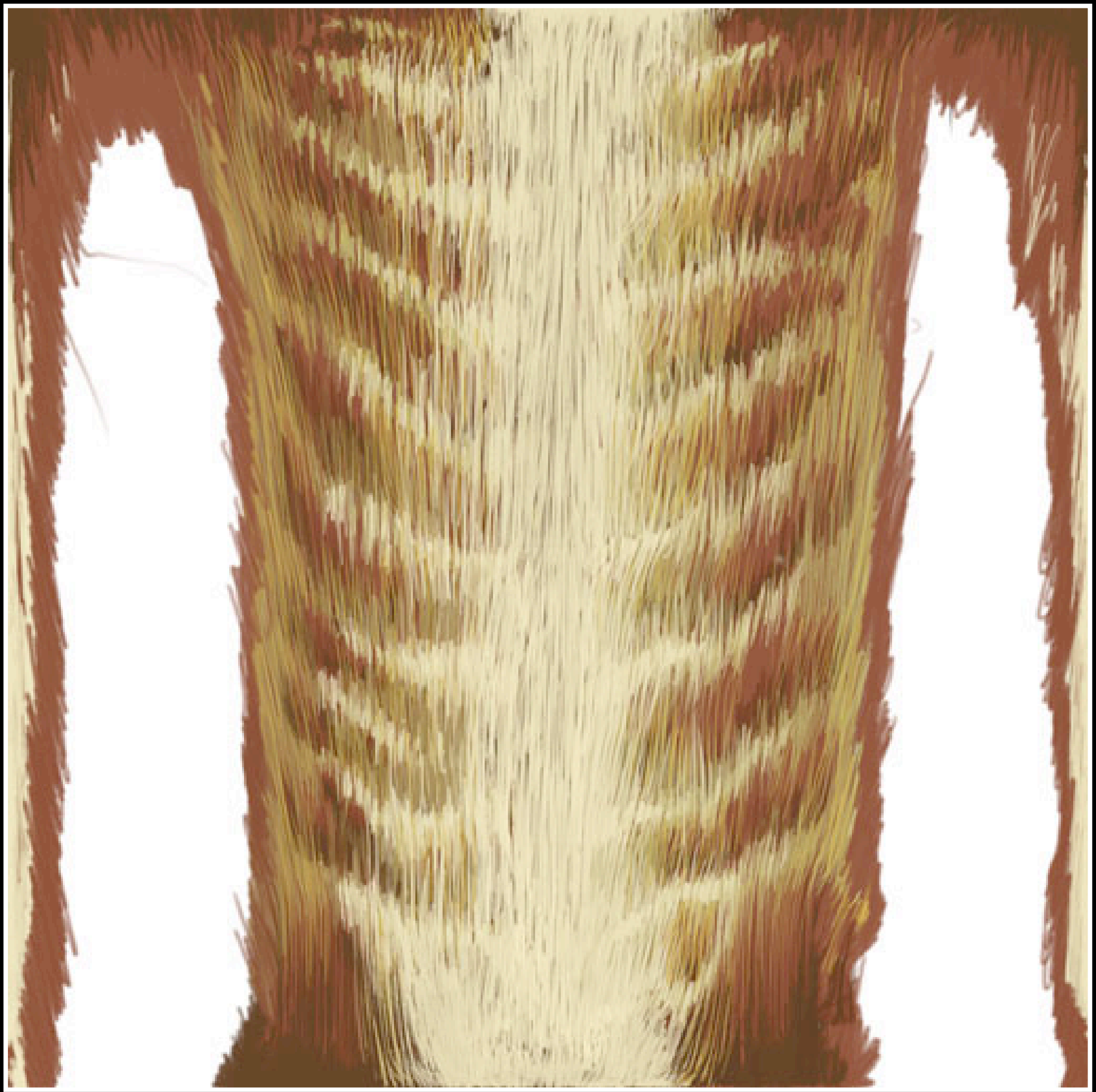
- We used 256 x 256 pixel fur textures, tiled over each surface
- Alpha component of texture represents density of fur
- RGB components represent tangent vector used for lighting



Fur Without Color Map



Fur Color Texture



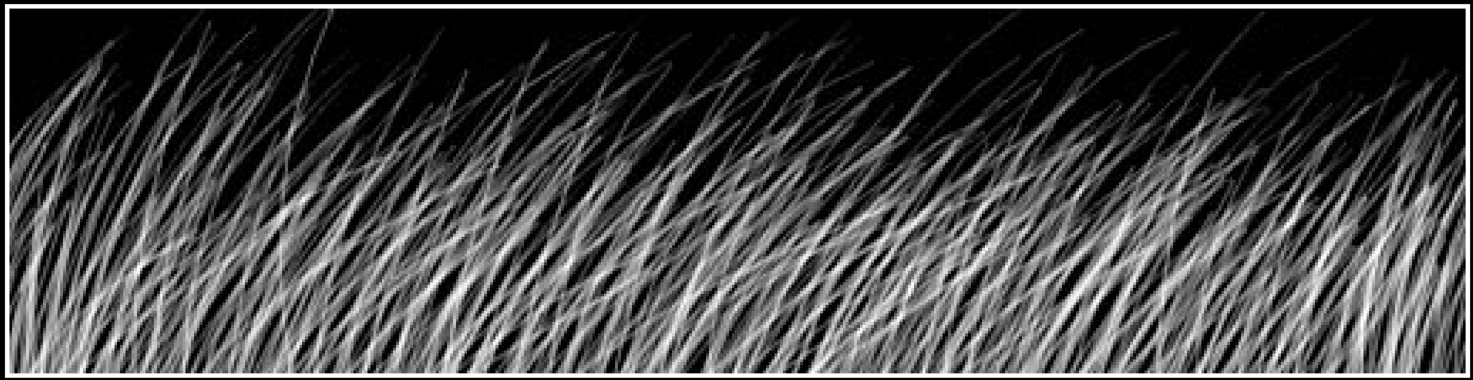
Fur With Color and Lighting

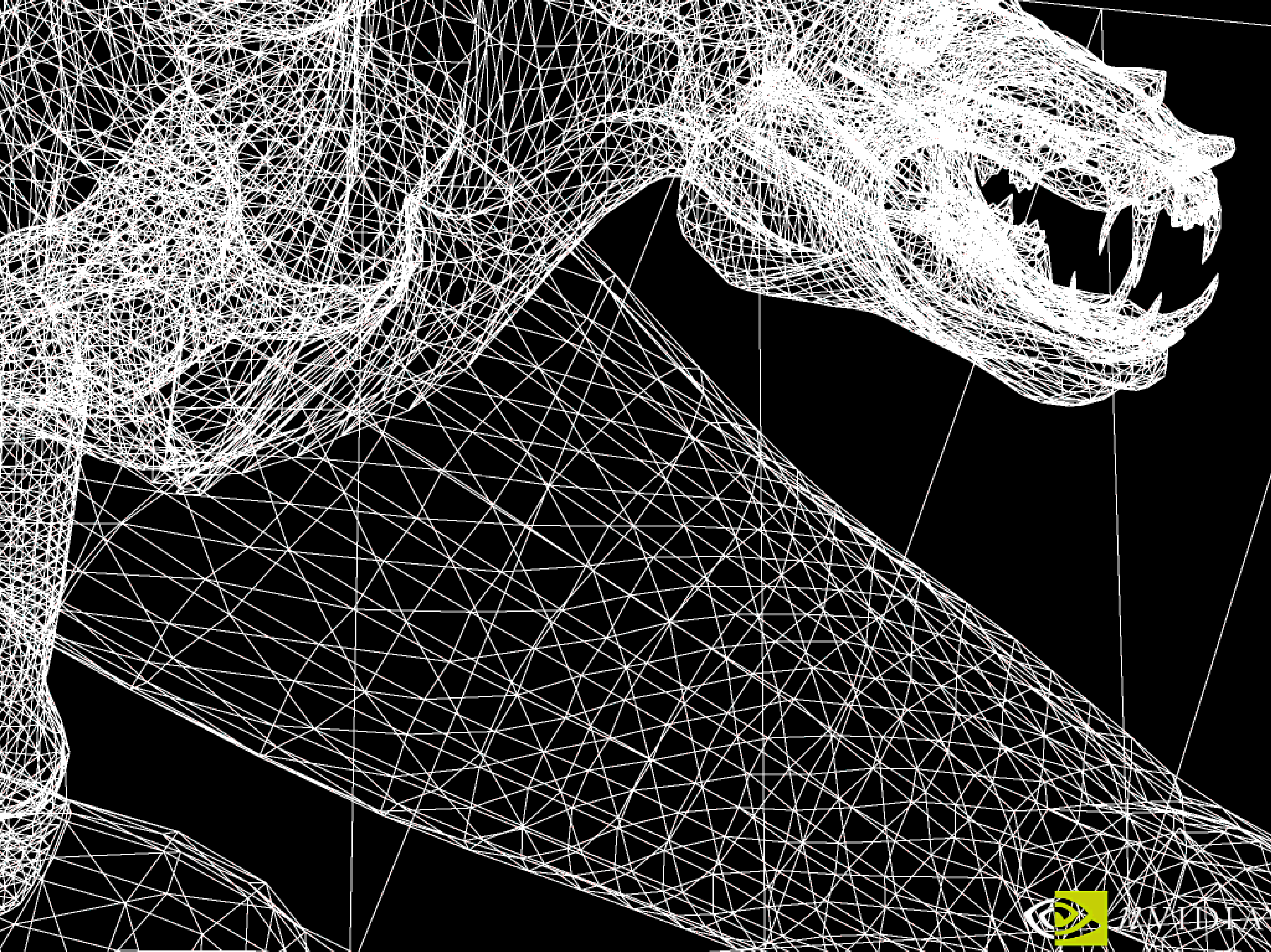
Rendering Fur With Shells and Fins

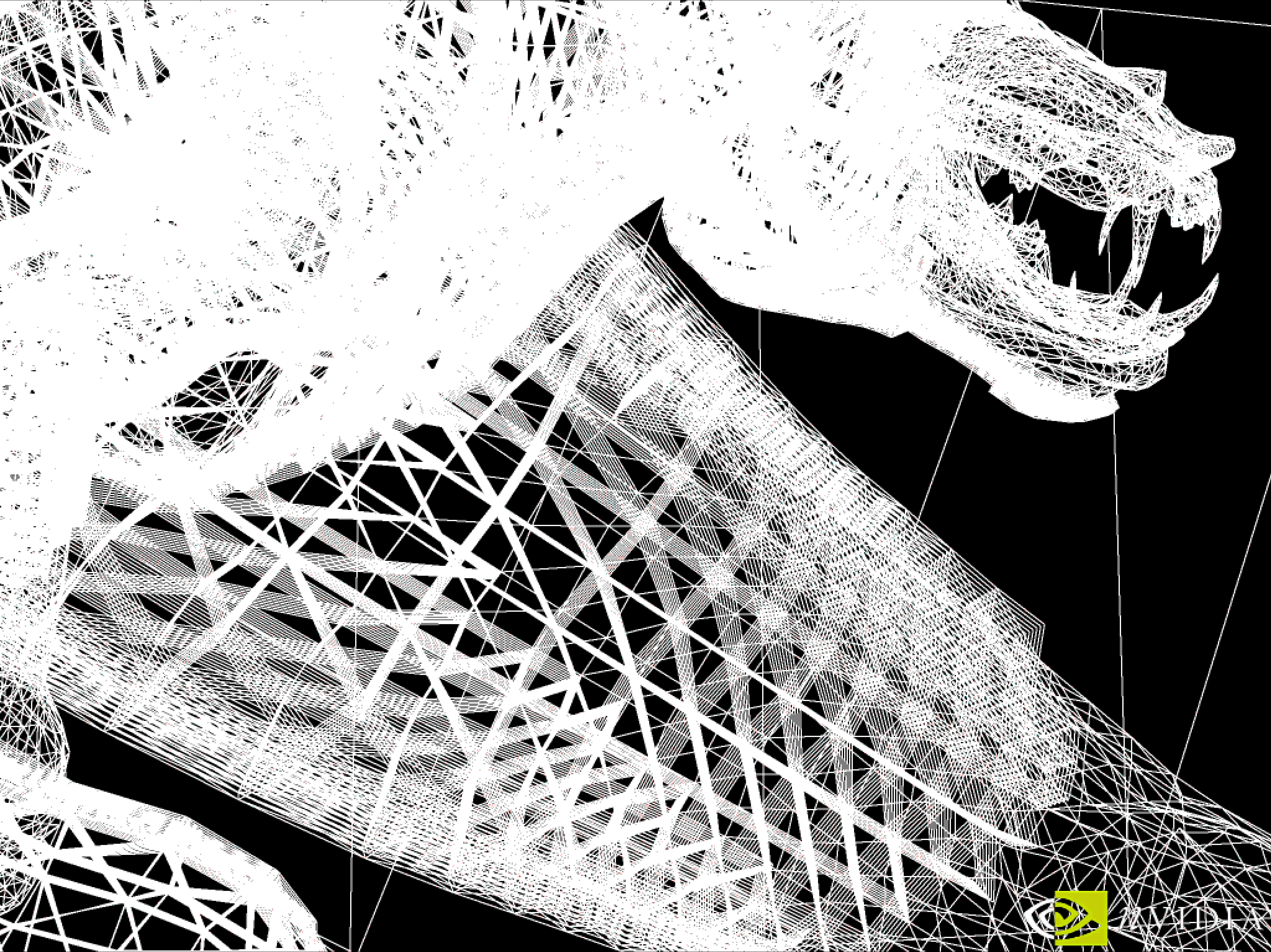
- Problem: fur using shells looks good, except on the silhouette
- Solution: add “fin” geometry to improve the edges
- Fins are generated by creating a quadrangle for each edge in the base mesh
- Textured with a separate image
- Fins are faded out based on the angle between the normal and the view direction, so that they only appear on the silhouette

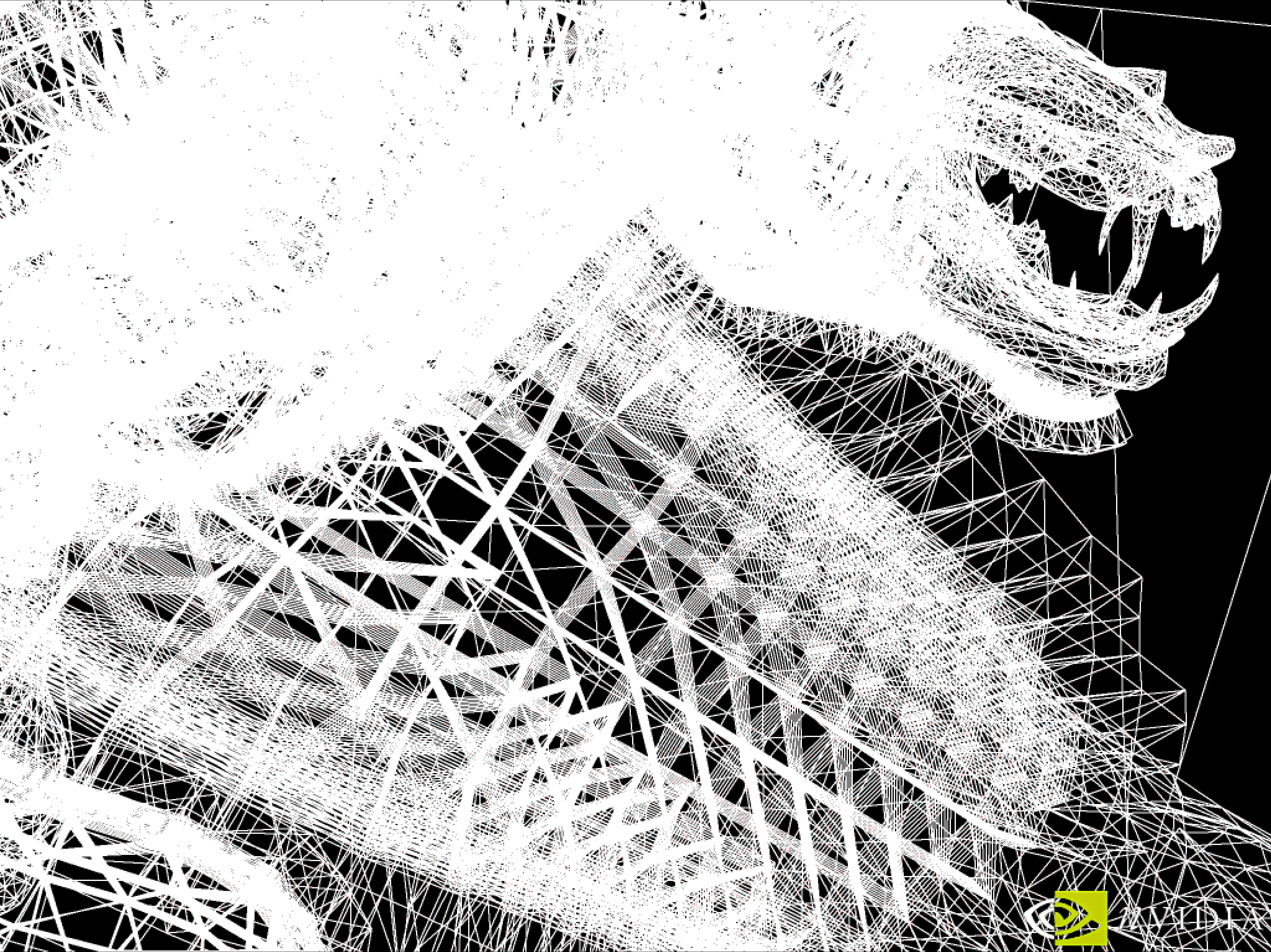
Fin Texture

- Hand painted, represents arbitrary cross section through fur:







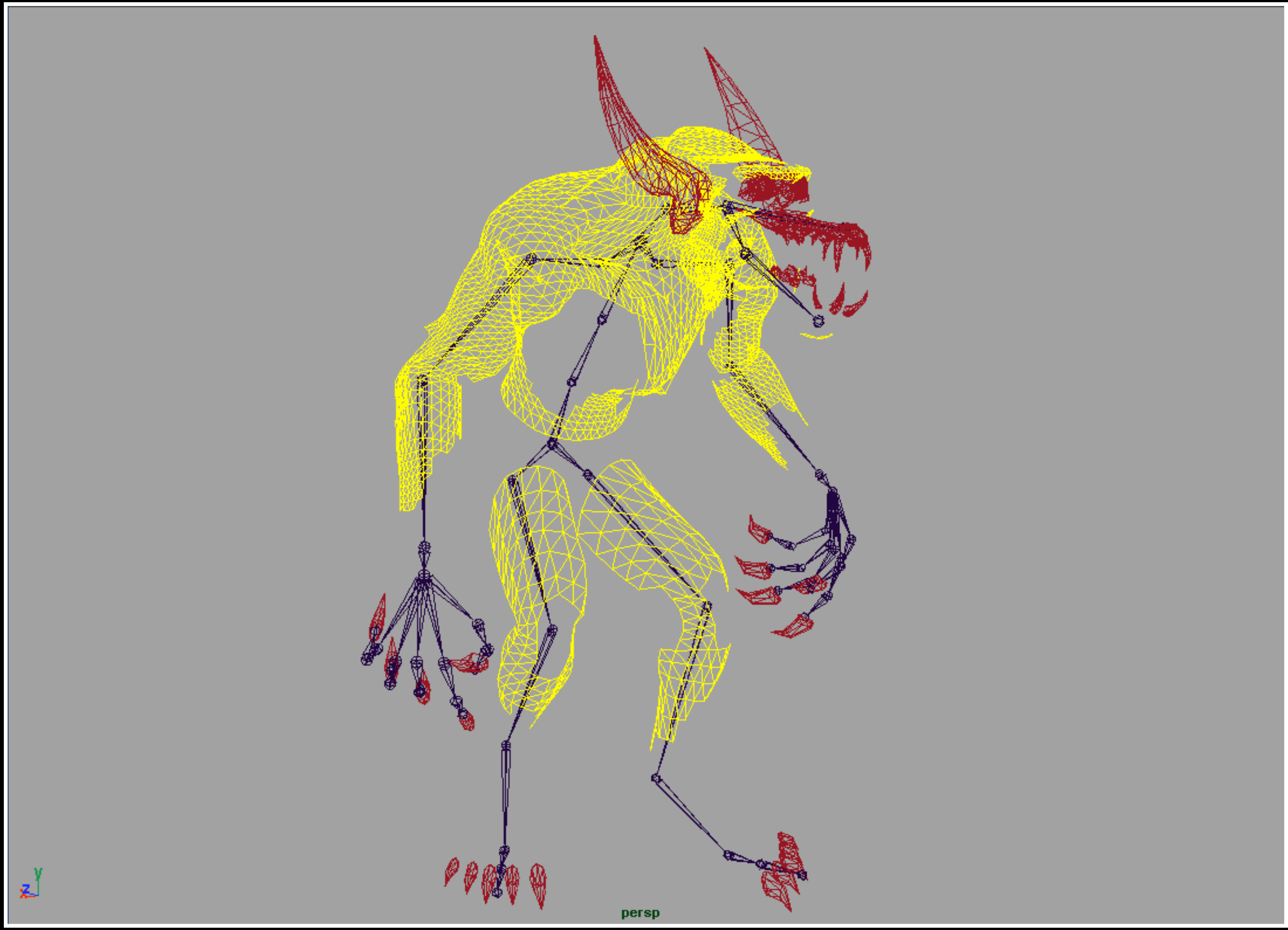



NVIDIA

Modeling and Animation

- Modeled in Maya as NURB surfaces
- Converted to polygons
- Base mesh has around 20,000 polygons total
- Skeleton has 61 bones
    - Including all fingers and thumbs!
- Comparable to complexity used in film and television production
- 925 frames of key-framed animation @ 30fps

Wolfman in Maya 4.0



NVDemo—The NVIDIA Demo Engine

- Proprietary NVIDIA demo engine
- Used for most of NVIDIA's in-house demos
- Object-oriented scene graph library
- Takes care of scene management, culling, sorting
- Includes Maya plug-in to convert geometry, materials, lights and animation to our own custom file format

Vertex Shaders

- Allow total control over hardware vertex processing using assembly language interface
- Exposed in OpenGL as **ARB_vertex_program** extension
- Used in Wolfman demo for:
  - Skinned animation
  - Scaling fur layers along vertex normal
  - Setup for per-pixel lighting
  - Texture coordinate generation for shadow mapping

Vertex Program Extract

```
// Load the index
ARL    A0.x,     v[V_INDEX].x;
// transform vertex by bone
DP4    R0.x,     c[A0.x + C_BONE0_X], v[V_COORD];
DP4    R0.y,     c[A0.x + C_BONE0_Y], v[V_COORD];
DP4    R0.z,     c[A0.x + C_BONE0_Z], v[V_COORD];
// multiply by weight
MUL    Rv.xyz,   R0,                  v[V_WEIGHT].x;

// transform normal by bone
DP3    R0.x,     c[A0.x + C_BONE0_X], v[V_NORMAL];
DP3    R0.y,     c[A0.x + C_BONE0_Y], v[V_NORMAL];
DP3    R0.z,     c[A0.x + C_BONE0_Z], v[V_NORMAL];
// multiply by weight
MUL    Rn.xyz,   R0,                  v[V_WEIGHT].x;
// repeat for each bone, plus binormals
```

Vertex Program Extract

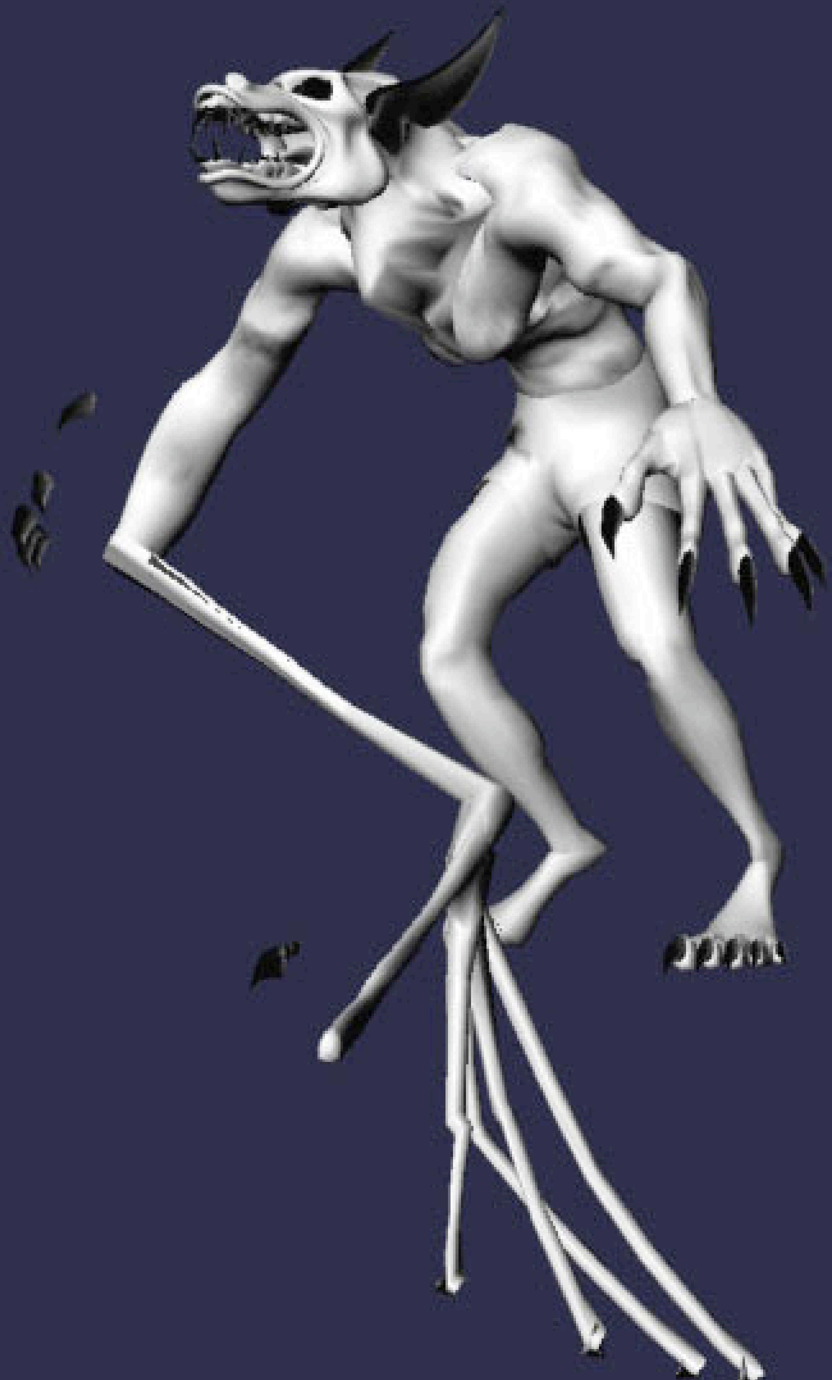
```
// scale vertex along normal
MAD   Rv.xyz, Rn, c[C_FUR_SHELL_SCALE].x, Rv;

// project vertex
DP4   o[HPOS].x,  c[C_PROJECTION_X],  Rv;
DP4   o[HPOS].y,  c[C_PROJECTION_Y],  Rv;
DP4   o[HPOS].z,  c[C_PROJECTION_Z],  Rv;
DP4   o[HPOS].w,  c[C_PROJECTION_W],  Rv;

// fur lighting
// calculate eye space view vector
DP3   Re.w, Rv, Rv;
RSQ   Re.w, Re.w;
MUL   Re.xyz, Rv, Re.w;
```

Vertex Program Extract

```
// calculate eye space half-angle vector
ADD    Rh, -Re, c[C_LIGHT0_DIRECTION];
DP3    Rh.w, Rh, Rh;
RSQ    Rh.w, Rh.w;
MUL    Rh.xyz, Rh, Rh.w;

// transform half-angle into tangent space
DP3    R0.x, -Rt, Rh;
DP3    R0.y, -Rb, Rh;
DP3    R0.z, Rn, Rh;

// map H into [0,1] and put into secondary color
MAD    o[COL1].xyz, R0, c[C_CONSTANTS].y, c[C_CONSTANTS].y;

// put fur diffuse lighting with attenuation into primary color
DP3    R0, Rn, c[C_LIGHT0_DIRECTION];
MUL    o[COL0].xyz, c[C_FUR_SHELL_SCALE].y, R0;
```

Skinning

- We want a smooth skin that covers the character and deforms as it animates
- Storing all the vertices for each key frame would be expensive
- Instead we animate a hierarchical skeleton, and use that to deform the skin
- Transform each vertex by multiple transformations—one for each of the nearby bones
- Final vertex position is a weighted average of the results of each of these transformations
- Weights are stored with each vertex
- Debugging skinning code can be fun . . .



NVIDIA


nVIDIA

Pixel Shaders

- Allow precise control over per-pixel operations
- Exposed in OpenGL as **NV_texture_shader** and **NV_register_combiners** extensions
- Used in Wolfman demo for:
  - Bump mapping (street and Wolfman's skin)
  - Anisotropic lighting model on fur
  - Shadows

Shadows

- Everything in the demo is shadowed using shadow maps
- Shadow maps are a two pass image-space technique
- Exposed via **GL_ARB_shadow extension**
- Advantages
    - Performance is linear with complexity of scene
    - Easy to implement
- Disadvantages
    - Aliasing
    - Shadow bias

Shadow Map Algorithm

- Render scene from light's point of view
- Copy depth information to "shadow map" texture
- Project shadow map texture back onto scene
- For each pixel, hardware compares depth in shadow map with depth of pixel:
    - If it's less, there must be something between us and the light shadowed
    - If it's roughly equal, point is visible from light

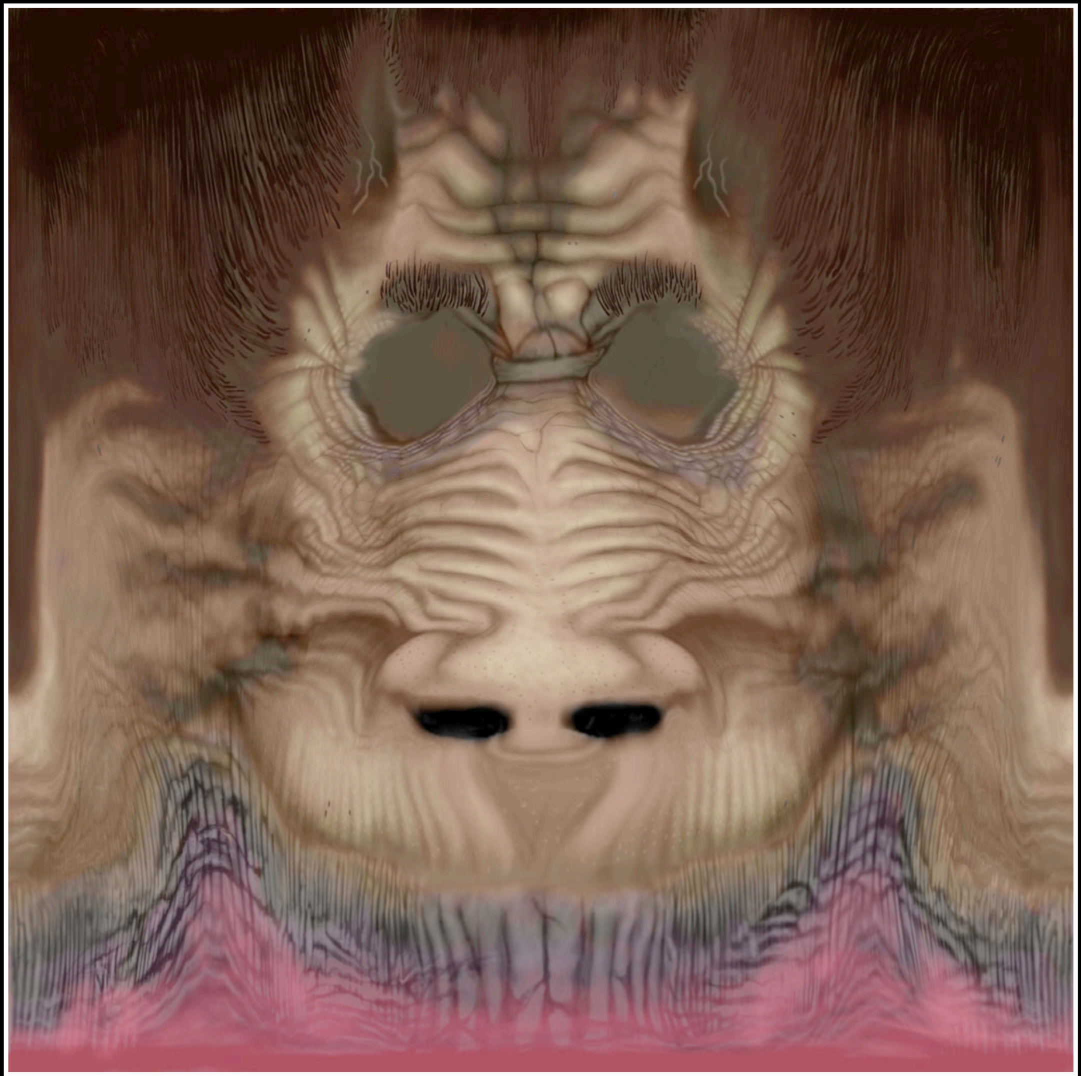
Bump Map Pixel Shader Pseudo Code

```
// NV_register_combiner pseudo code for bump mapping
texture0:       color map (alpha = shininess map)
texture1:       bump map
texture2:       Phong specular map
texture3:       shadow map
primary:  light direction (L)
secondary:      half angle vector (H)

// diffuse lighting = N.L
reg0.rgb = primary . texture1
// calculate shadow factor
reg1.alpha = texture3*(1.0 - factor) + factor;
// diffuse * color map
reg0.rgb = reg0 * texture0
```

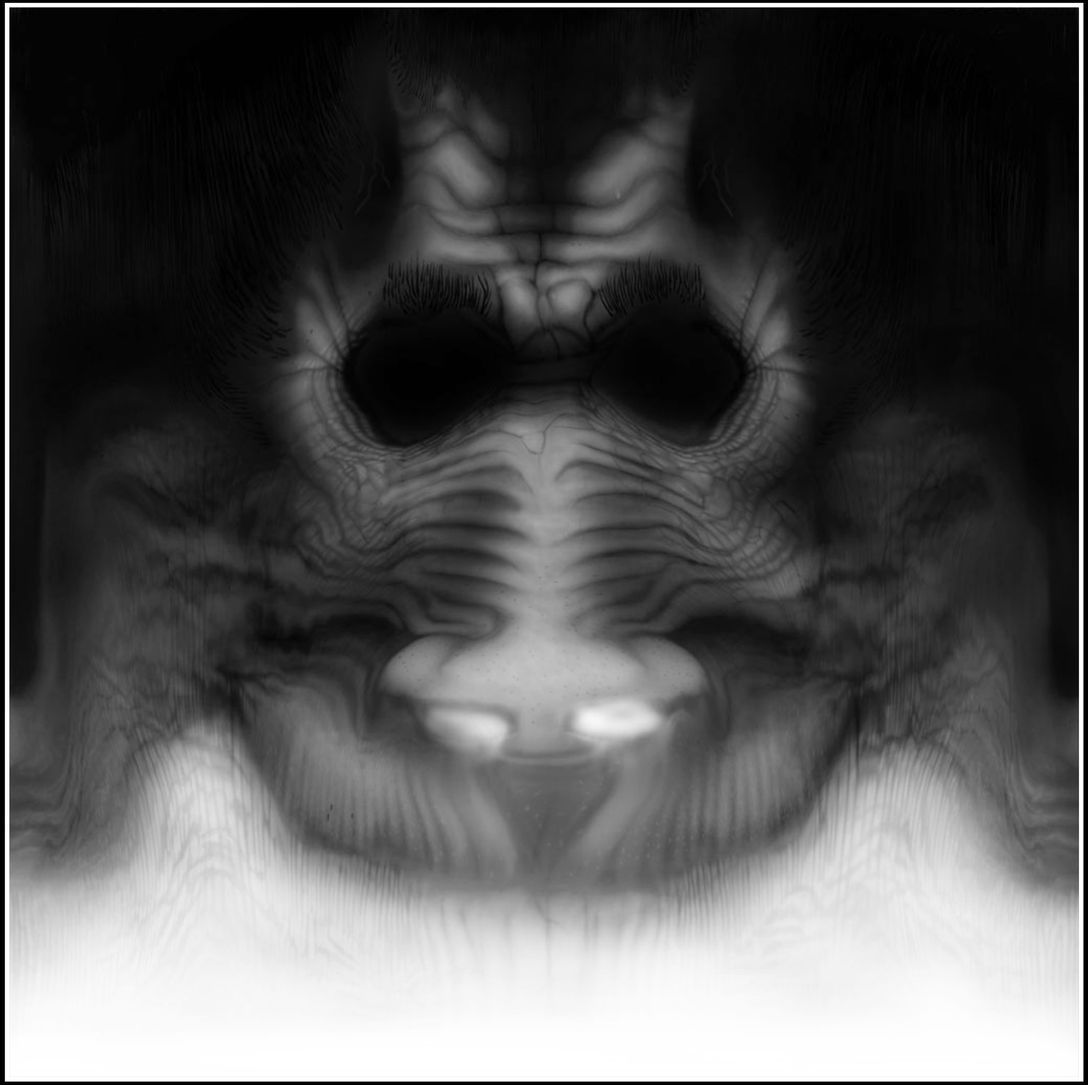
Bump Map Pixel Shader Pseudo Code

```
// specular * shininess
reg0.alpha = texture2 * texture0.alpha

// shadowing
// diffuse*shadow_factor + specular*shadow
reg0.rgb = reg0*reg1.alpha + reg0.alpha*texture3

// fog
reg0.rgb = interp(reg0, fog_color, fog_factor);

// NV_texture_shader psuedo code
texture_2D();
texture_2D();
dot_product_texture_1D(tex1);    // computes (N.H)^p
texture_2D();
```

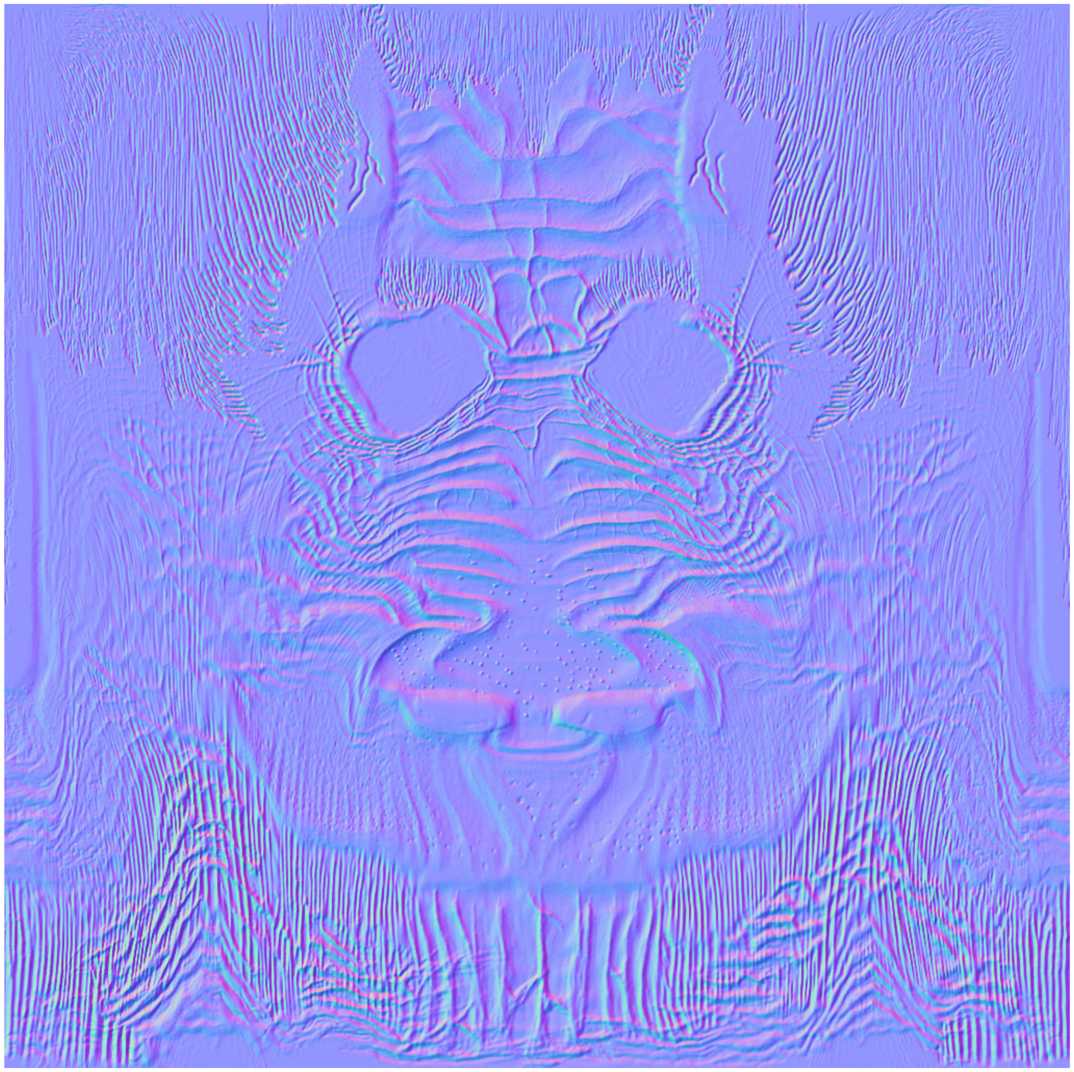
Texture–Color Map



Texture–Shininess Map



Texture–Normal (Bump) Map



Final Result

Future Work

- Fur dynamics
  - Animate shell geometry independently from model based on physical simulation
- Ray marching
  - Blend layers inside pixel shader
  - Produce illusion of depth without shells
  - Reduces frame buffer blending
- Geometry-based fur
  - Antialiasing and shadowing are hard

Summary

- Programmable Vertex and Pixel shaders allow *you* to control the hardware
- Real-time and offline production rendering are converging
- Programmable graphics hardware acceleration is applicable not only to games, but 2D imaging, video processing and user interfaces . . .
- Next generation hardware will be faster, and even more programmable
- Start learning this stuff now!

Demo Credits

- Curtis Beeson, Joe Demers—Engine Code
- Daniel Hornick—Modelling and Texturing
- Jeff Bell—Character Animation
- Ken Kurita-Ditz—Sound Design
- Simon Green—Additional Code and Shaders
- Mark Daly—Producer


nVIDIA

References

- J. Kajiya and T. Kay, *Rendering fur with three dimensional textures*, SIGGRAPH Proceedings, pp. 271–280, 1989
- Jerome Lengyel, Emil Praun, Adam Finkelstein, Hugues Hoppe, *Real-Time Fur over Arbitrary Surfaces*, ACM 2001 Symposium on Interactive 3D Graphics

http://research.microsoft.com/~jedl/

http://developer.nvidia.com/view.asp?IO=nvidia_opengl_specs

Roadmap

514 OpenGL: Performance and Optimization	Room J **Thur., 5:00pm**
FF018 Graphics and Imaging	Room J1 **Fri., 5:00pm**

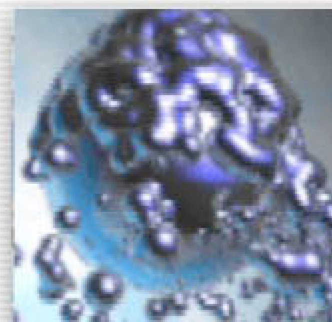
Who to Contact

Simon Green
NVIDIA Demo Engineer
sgreen@nvidia.com

Sergio Mello
3D Graphics Technology Manager
sergio@apple.com

http://developer.apple.com/wwdc2002/urls.html

Q&A



Sergio Mello
3D Graphics Technology Manager
Worldwide Developer Relations
sergio@apple.com

http://developer.apple.com/wwdc2002/urls.html

WWDC2002

WWDC2002

WWDC 2002